CS2102 Notes

github.com/reidenong/cheatsheets, AY25/26 Sem 1.

Introduction & Relational Model

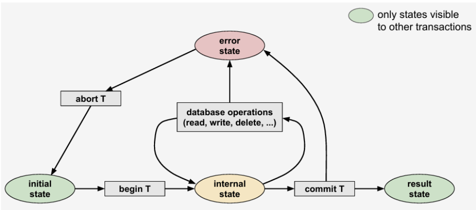
Transactions

Transactions, T are finite sequences of database operations.

ACID

- **Atomicity**: either all effects of T are reflected in the database or none
- **Consistency**: the execution of T guarantees to yield a correct state of the database
- **Isolation**: the execution of T is isolated from the effects of concurrent Transactions
- **Durability**: once T has been committed, its effects are permanent even in case of failures.

Transition Graph of a Transaction



Errors of Concurrent Execution

- **Lost update**: Read-read-write-write, lose the latest written
- **Dirty read**: Transactions are interleaved, we want to abort the first transaction but its results are already in the second transaction and we end up with a faulty result.
- **Unrepeatable read**: After the first read of the first transaction, we have a second transaction committed, and the first transaction has a different value.

Architecture of DBMS

1. User/group-specific view of the data (External Schema)
2. Logical schema: Logical organization of the data
3. Physical schema: Organization of data on disk and in memory

Querying

We write SQL queries, which go through the following steps:

1. Parser does parsing and validation to form a relational algebra (RA) expression
2. This then goes through a query optimizer to form a query execution plan
3. The query execution plan then goes through a code generator to get executable code

Data independence

- Logical data independence is the ability to change logical schema without affecting external schemas (shielding the applications from changes in the logical structure of the data).
- Physical data independence is the ability to have representation of data independent from physical schema (shielding the logical structure from changes in the storage and access models).

Relational Data Model

A domain refers to the set of atomic values that a column can take. $\text{dom}(A_i)$ = set of possible values of A_i. It follows that each value v of attribute A_i must satisfy the condition $v \in \text{dom}(A_i)$.

A relation refers to a set of tuples. $R(A_1, A_2, ..., A_n)$ is a relation schema with name R and attributes $A_1, A_2, ..., A_n$. Each instance of schema R is a relation which is a subset of $\{(a_1, a_2, ..., a_n) \mid a_i \in \text{dom}(A_i) \cup \{\text{null}\}\}$

Integrity Constraints

The DBMS checks that tables only ever contain valid data. There are 3 main structural integrity constraints of the relation Model:

1. Domain Constraints
2. Key Constraints
3. Foreign Key Constraints

Terminology

- **Attribute**: A column in a table.
- **Domain**: The set of all possible values an attribute can take.
- **Attribute value**: A element from an attribute's domain.
- **Relation schema**: A set of attributes that defines the structure of a relation (table).
- **Tuple**: A single row (record) in a table.
- **Relation**: A set of tuples sharing the same relation schema (i.e., a table).
- **Database schema**: The set of all relation schemas in a database.
- **Database**: A set of relations (tables) conforming to a database schema.
- **Superkey**: A set of one or more attributes that uniquely identifies a tuple in a relation.
- **Key**: A minimal superkey (i.e., no proper subset of it is a superkey).
- **Candidate key**: The set of all keys for a relation.
- **Primary key**: The chosen candidate key used to uniquely identify tuples. Cannot be null and must be unique.
- **Foreign key**: An attribute (or set of attributes) in one relation that refers to the primary key of another relation.

Foreign Key Constraints

To ensure that all references are valid and pointing to existing entities, each foreign key value in referencing relation must

- Appear as primary key in referenced relation

OR

- Be a null value

Integrity Constraints Limitations

- Structural integrity constraints do not cover application-independent constraints (eg. limiting the domain to valid values)
- Integrity constraints are optional
- Integrity constraints may affect performance

Entity Relationship Model

- Data can be described in terms of entities and relationships.
- Information about entities and relationships are described using attributes.

Definitions

- **Entity**: Real world things or objects
- **Entity Set**: Collection of entities of the same type, usually named as nouns (rectangle)
- **Attribute**: Specific information describing an entity
  - **Key attribute**: uniquely identifies each entity (filled circle)
  - **Derived attribute**: derived from other attributes (dashed line)
  - **Composite key attributes**: attributes that together uniquely identify each entity
  - Multivalued attributes: An attribute may refer to a set/list of values
- **Relationship**: Association between two or more entities
- **Relationship set**: Collection of relationships of the same type, which may have their own attributes that further describe the relationship, typically named as verbs. (diamond)
- **Role**: description of an entity's participation in a relationship
- **Degree**: Number of entity roles in a relationship set

Cardinalities

- Cardinality: Describes how often an entity can participate in a relationship at most
  - many to many
  - many to one
  - one to one
- Participation constraints: describe how often an entity has to participate in a relationship at least.
- Described in (min_num, max_num)

Structural Constraints

Cardinality Constraints

- Enforcing cardinality constraint on A with respect to B
  - Enforcing how many B records each A might have

Total Participation Constraints

- Enforcing total participation constraint on A with respect to B
  - Every A must appear in B at least once

Covering Constraints

- States whether every entity in the superclass must appear in at least one of its subclasses
  - Total Covering: Every superclass must belong to at least 1 subclass
  - Partial Covering: Some superclass may belong to no subclass

Overlap Constraints

- States whether an entity of the superclass may appear in multiple subclasses simultaneously
  - Overlapping: An entity may belong to more than one subclass at once
  - Disjoint: An entity may belong to at most one subclass at a time

Dependency Constraints

- Weak entity sets: An entity set that cannot exist without an owner, without its own key. It has a partial key that depends on the on its owner instance. Must have a (1,1) attachment to identifying relationship

Relational Algebra

- Relations are closed under relational algebra, ie. the all input operands and the output of all operators are relations
- This property allows for the nesting of relational operators

Operators

Renaming, ρ

- Renaming relation: $\rho(R, S)$
- Renaming attributes: $\rho(R, R(a_1 \to b_1, ...))$
- Renaming both: $\rho(R, S(a_1 \to b_1, ...))$

Selection, σ_c

- $\sigma_{c(R)}$ selects all tuples from relation R that satisfy selection condition c (ie. evaluates to **true**)

Selection Conditions

- Constant selection: c = attribute **op** constant
- Attribute selection: c = attribute **op** attribute
- Expression manipulators: $\wedge, \vee, \neg, ()$
- Operators: $=, <>, <, \leq$
- Precedence: $()$, op, $\neg, \wedge, \vee$

Projection, π_l

- $\pi_{l(R)}$ projects all the attributes of R specified in list l
- Since relation refers to the set of tuples, π removes duplicate tuples from output relation

Set Operators

- Union $\cup$, Intersection $\cap$, Difference $-$
- Two relations R, S are union-compatible if they have the same number of attributes which have the same or compatible domains
  - Do not have to use the same attribute names

Cross Product, $\times$

- $\times$ combines two relations by forming all pairs of tuples
- $|R \times S| = |R| \cdot |S|$

Inner Join, $\bowtie_\theta$

$$R \underset{\theta}{\bowtie} S = \sigma_\theta(R \times S)$$

where $\theta \in \{=, <>, <, \leq\}$

Natural Join, $\bowtie$

- Behaves like $\bowtie_=$ but is perfomed on all common attributes of two relations R, S (ie. same name)

$$R \bowtie S = \pi_l\left(R \underset{c}{\bowtie} \rho_{b_i \leftarrow a_i, ...}(S)\right)$$

where

- $A = \{a_i, ...a_k\}$ is the set of attributes common to R, S
- $c = (a_i = b_i) \wedge ... \wedge (a_k = b_k)$
- l is the list of all attributes of R + list of all attibutes in S not in R

Outer Joins

- Outer joins preserve tuples that do not match with tuples in the other relation (dangling tuples), padding them with nulls
- Left outer join ⟕: R ⟕ S with dangling tuples from R
- Right outer join ⟖: R ⟖ S with dangling tuples from S
- Full outer join ⟗: R ⟗ S with dangling tuples from R, S

- $dangle(R \bowtie_\theta S)$ is the set of dangling tuples in R w.r.t $R \bowtie_\theta S$.
- $null(R)$ is the n-component tuple of null values where n is the number of attributes of R
  - eg. if R has 2 attributes, $null(R)$ = (*null, null*)

Then,

$$R \underset{\theta}{⟕} S = R \underset{\theta}{\bowtie} S \cup \left(dangle\left(R \underset{\theta}{\bowtie} S \right) \times \{null(S)\} \right)$$

and

$$R \underset{\theta}{⟖} S = R \underset{\theta}{⟖} S \cup \left(\{null(R)\} \times dangle\left(R \underset{\theta}{\bowtie} S \right) \right)$$

Natural Outer Join, ⟗

- Only the equality operator is used, performs join over all attributes of R, S in common

Invalid Relational Expressions

- Attribute no longer available after Projection
- Attribute no longer available after Renaming
- Incompatible attribute types

Stored Procedures and Functions

- Stored procedures (has no return) and functions (has return) allow for creation and execution of code directly within the database
- Pros: Allows for implementation and maintenance of code in a single place, and minimizes network latency vs. having to write client side code
- Cons: Does not benefit from optimization by DBMS, and the code is not portable

Triggers

- A trigger is a procedure or function executed when a databse event occurs on a table
- `BEFORE` Triggers activate before a database event, use to enforce constraints, validate incoming data, block operations, and set fields.
  - `RETURN NEW` : allow `INSERT` / `UPDATE`
  - `RETURN OLD` : allow `DELETE`
  - `RETURN NULL` : Cancel operation
  - `NEW` / `OLD` refer to what the row looks like after/before the operation. It may not always be available.
- `AFTER` Triggers are used to update related tables, send notifications and to update derived tables. Return value is ignored.

Anomalies, Functional Dependencies

Functional Dependencies

- Some columns are uniquely determined by other columns
- An instance r of relation schema R satisfies the functional dependencies σ of the form $X \to Y$ with $X \subseteq R$ and $Y \subseteq R$ if and only if two tuples of r agree on their X-values, then they agree on their Y-values
- $X \to Y$: X (functionally) determines Y

Set Functional Dependency

- An instance r of a relation schema R satisfies a set of functional dependencies Σ if and only if it satisfies all the functional dependencies $\sigma \in \Sigma$.
- We say that a set of functional dependencies Σ holds on a relation R

Triviality

- A functional dependency $\sigma : X \to Y$ is trivial if and only if $Y \subseteq X$
- A functional dependency $\sigma : x \to Y$ is completely non-trivial if and only if $Y \neq \emptyset$ and $Y \cap X = \emptyset$
  - A non-trivial functional dependency can be split into a trivial and completely non-trivial functional dependency.

Superkey

- A superkey is a set of attributes of a relation whose values determine the value of the entire tuple
  - Formally, for relation R, $S \subseteq R$, S is a superkey of R if and only if $S \to R$

Candidate Key

- A candidate key of schema R is a minimal superkey, where if any attribute is removed it is no longer a superkey
  - Formally, for relation R, for S to be a candidate key for $R, \forall T \subset S$, T is not a superkey of R
  - Set of all candidate keys are unique

Finding Candidate Key

- To calculate candidate key, for all $X \to Y \in \Sigma$, check the attribute closure of $X \cup Z$ where Z is the set of all attributes
  - In the closures, if any attribute does not appear on the RHS $\forall \sigma \in \Sigma$, it must be part of all candidate keys
  - Compute closures of all singletons, then pairs, and so on

Prime Attributes

- An attribute that appears in some candidate key of R with Σ. If there are multiple candidate keys K_i, then the set of all prime attributes is $\cup_i K_i$

Closures

- Let Σ be the set of functional dependencies of schema R. Then Σ^+ is the closure of Σ, the set of all functional dependencies logically entailed by the functional dependencies in Σ.
- The closure of $S \subseteq R$ is S^+ and is the set of all attributes functionally dependent on S.
  - We can determine this as a graph problem, where we want to find all nodes reachable from S.
  - Note that for a functional dependency $X \to Y$, Y is only reachable if we can reach everything in X.
- Two sets of functional dependencies are equal if they have the same closure, ie. $\Sigma_1 \equiv \Sigma_2 \Leftrightarrow \Sigma_1^+ = \Sigma_2^+$

Covers

- A set Σ of functional dependencies is minimal if and only if
  - The RHS of every functional dependency S is minimal (ie. one attribute)
  - The LHS of every functional dependency S is minimal (a minimal set)
  - The set Σ itself is minimal, ie. none of the functional dependencies inside can be derived from other functional dependencies.
- We can produce a canonical cover of Σ by taking a minimal cover Σ' and regrouping all the functional dependencies with the same left hand side in Σ_1

Normal Forms

BCNF $\subset$ 3NF $\subset$ 2NF $\subset$ 1NF

Decomposition

- A binary decomposition is lossless-join if and only if the natural join of its two fragments equals the initial table
  - Binary decomposition of $R \to R_1, R_2$ is also lossless if $R = R_1 \cup R_2$ and $R_1 \cap R_2 \to R_1$ or $R_1 \cap R_2 \to R_2$.
- A decomposition of R is lossless-join if there exists at least one sequence of binary lossless-join decomposition that generates that decomposition

Determining if a decomposition is lossless-join

- Consider all binary splits of $R \to R_1, ..., R_k$; if there exists a sequence of binary decompositions where each split is not lossless then the decomposition is lossless
- For each split:
  1. For R_1, R_2, find the intersect $R_1 \cap R_2$
  2. Find $A = \{R_1 \cap R_2\}^+$ using the functional dependencies
  3. if $A = R_1$ or $A = R_2$ then it is lossless
     - ie. A is a superkey for R_1 or R_2

Projected Functional Dependencies

- With schema R, for $R' \subseteq R$, the projection of Σ on R, $\Sigma|_{R'}$, is the set of all functional dependenciees $X \to Y$ where $X \subseteq R'$ and $Y \subseteq R'$

Finding projection of Σ in R'

1. List all subsets of attributes in R'
2. Compute each of their attribute Closures (which may include attributes not in R')
3. From all RHS, remove attributes not in R'
4. From RHS, remove all attributes from LHS

Prof Adi's Fast-but-unproven method

1. Take all $\sigma \in \Sigma$ whose LHS $\subseteq R'$ and place in A.
2. $\forall (X \to Y) \in A$, compute closures of X
3. Then for each X^+, remove attributes not in R' and all trivial attributes, and we are done

Dependency-Preserving Decomposition

- A decomposition of R, Σ into $\{R_1, ..., R_n\}$ is dependency preserving if and only if $\Sigma^+ = \left(\Sigma|_{R_1} \cup ... \cup \Sigma|_{R_n}\right)^+$

Checking if decomposition is dependency preserving

- Find $\Sigma_\cup = \left(\Sigma|_{R_1} \cup ... \cup \Sigma|_{R_n}\right)^+$
- Check each functional dependency $X \to Y \in \Sigma$. Compute each X^+ w.r.t. $\Sigma_\cup$. If $Y \nsubseteq X^+$ from $\Sigma_\cup$ then the decomposition is not dependency preserving.

Boyce-Codd Normal Form

- A relation R with functional dependencies Σ is in BCNF if and only if for every functional dependency $X \to \{A\} \in \Sigma^+$:
  - $X \to \{A\}$ is trivial
  - X is a Superkey

BCNF Decomposition

- Given R, Σ, find a $X \to Y \in \Sigma$ that violates the BCNF definition (ie. non trivial *and* X not a superkey)
- Let $R_1 = X^+$, $R_2 = (R - X^+) \cup X$
- Recurse on $\left(R_1, \Sigma|_{R_1}\right), \left(R_2, \Sigma|_{R_2}\right)$ and return their union.

3NF

- A relation R with functional depenencies Σ is in 3NF if and only if for every functional dependency $X \to \{A\} \in \Sigma^+$:
  - $X \to \{A\}$ is trivial
  - X is a Superkey
  - A is a prime attribute

3NF Synthesis (Bernstein algorithm)

- Lossless-join and dependency preserving

1. Compute candidate keys
2. Compute minimal cover Σ_C of Σ
3. Compute canonical cover Σ_D
4. Synthesize R_i for each $\sigma \in \Sigma_D$
5. Remove subsumed relations
6. Add back candidate keys as additional R_i if needed